On the structure of problem variability: From feature diagrams to problem frames

Conference or Workshop Item

On the Structure of Problem Variability: From Feature Diagrams to Problem Frames

Andreas Classen*, Patrick Heymans
Computer Science Department, University of Namur
5000 Namur, Belgium
{aclassen, phe}@info.fundp.ac.be

Robin Laney, Bashar Nuseibeh, Thein Than Tun
Centre for Research in Computing, The Open University
Walton Hall, Milton Keynes MK7 6AA, UK
{r.c.laney, b.nuseibeh, t.t.tun}@open.ac.uk

Abstract

Requirements for product families are expressed in terms of commonality and variability. This distinction allows early identification of an appropriate software architecture and opportunities for software reuse. Feature diagrams provide intuitive notations and techniques for representing requirements in product line development. In this paper, we observe that feature diagrams tend to obfuscate three important descriptions: requirements, domain properties and specifications. As a result, feature diagrams do not adequately capture the problem structures that underlie variability, and inform the solution structures of their complexity. With its emphasis on separation of the three descriptions, the problem frames approach provides a conceptual framework for a more detailed analysis of variability and its structure. With illustrations from an example, we demonstrate how problem frames analysis of variability can augment feature diagrams.

1 Introduction

A *software product line* (SPL) is *"a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* [6]. *Software Product Line Engineering* (SPLE) is a rapidly emerging software engineering paradigm that institutionalises reuse throughout software development. By adopting SPLE, one expects to benefit from economies of scale and thereby improving the cost, productivity, time to market, and quality of developing software.

Central to the SPLE paradigm is the modelling and management of *variability*, i.e. "*the commonalities and differences in the applications in terms of requirements, architecture, components, and test artefacts*" [21]. In order to tackle the complexity of variability management, a number of supporting modelling languages have been proposed. In this paper we examine two approaches for analysing requirements for SPLs: *Feature Diagrams* and *the Problem Frames approach*.

Feature Diagrams (FD) [16, 17, 9, 8, 28, 2] are mostly used to model the variability of application "features" at a relatively high level of granularity. Their main purposes are (i) to capture feature commonalities and variabilities, (ii) to represent dependencies between features, and (iii) to determine combinations of features that are allowed and disallowed in the SPL. An important limitation of FDs is that they tend to mix requirements, domain properties and specifications. For example, in FDs, it is not clear whether variability is a domain, requirement or solution property.

The Problem Frames approach (PF) by Jackson [14] is a more general approach to requirements engineering. This approach emphasises a clear distinction between requirements, domain descriptions and specifications. However, the PF approach has not been applied in the context of feature-based development.

In this paper we show that FDs and the PF approach can be used as complementary techniques. The basic idea is that a PF analysis would serve as an early requirements analysis in product line development.

Yu et al. [30] identify lack of organisational and motivational context in FDs and propose goal models to com-

*Currently visiting at The Open University, UK.

plement FD. Similarly, Halmans and Pohl [12] suggest that use case diagrams can help in communicating variability to different stakeholders. However, with the PF approach we aim to put FDs in the context of domains in the real world. Benefits of this approach include (i) existing FDs will be supplemented by an analysis of underlying problem complexity, (ii) notations and semantics of FDs and problem frames are not changed or extended, (iii) this approach is not prescriptive, meaning no change to process or methodology is suggested, and (iv) FDs can link problem frames to the solution space. We believe that our approach to use problem frames as an early analysis in SPL development is a novel contribution to this discussion.

The rest of this paper is organized as follows. Section 2 describes the background to this work by providing an overview of FDs and the PF approach. Section 3 introduces the illustrative example, which is then analysed using FDs in Section 3.1, and using the PF approach in Section 3.2, before revisiting the initial analysis in Section 3.3. Section 4 gives an overview of related work. Discussions and conclusions can be found in Section 5.

2 Background

In this section we will briefly review the two approaches considered in this paper, Feature Diagrams and the Problem Frames approach.

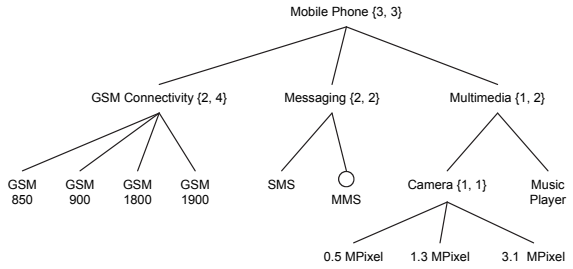
2.1 Feature Diagrams

Feature diagrams are a common notation for representing requirements of SPLs using a feature tree, or a directed acyclic graph. Since their introduction in 1990 as part of the FODA method [16], many new or adapted notations have been published. Schobbens et al. [27, 25] give an overview of the existing FD notations and approaches. They define a formal semantics for FD notations. They also introduce their own FD language, varied FDs (VFD) inspired by Riebisch et al. [23], which only contains one decomposition relation with cardinalities. Their notation is used throughout this paper[1].

Figure 1, adapted from [26], is an example of a simple feature diagram for a mobile phones product line. A mobile phone has a GSM connectivity feature, a messaging feature and a multimedia feature. They are linked by an *and*-relation, as the cardinality $\{3, 3\}$ of the top feature indicates, which means that they all have to be included in a product. The cardinality $\{2, 4\}$ of the GSM connectivity feature means that at least two frequency bands have to

[1]To be precise, Schobbens et al. only defined the semantics and an abstract syntax. The concrete syntax we use here is inspired by Riebisch et al.'s FORE FD. In our concrete syntax, unary $\{0, 1\}$ decompositions (i.e. optional features) are simplified with hollow circles.

Figure 1. A feature diagram for a mobile phone product line.



be supported. The messaging feature has two subfeatures connected by an *and*-relation: the mandatory SMS feature and the optional MMS feature (indicated by the hollow circle above it). It does not have to be included even though the cardinality of its parent node is $\{2, 2\}$. The Multimedia feature is decomposed with an *or*-relation (cardinality $\{1, 2\}$), that means that at least one of its subfeatures has to be included. The Camera feature is decomposed with a *xor*-relation (cardinality $\{1, 1\}$), meaning that only one of the three picture resolutions can be supported by the mobile phones' camera.

Features in FDs generally refer to requirements R, but they can also represent domain properties W, specifications S and design D, leading to confusion as to what exactly FDs are describing. This is reflected for example by the definitions by Kang et al. [17]: "*a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems*". This notion of "feature" is not restricted to R, but also includes S and even D. As a result, FDs often do not reveal much about the underlying complexity of problem structures.

FDs can also represent variability with different binding times, such as design-time variability and run-time variability according to van Gurp et al. [29]. The scope of this paper will be limited to design-time variability, that is variability occurring in the requirements. For a discussion on the PF approach to run-time variability, also referred to as context-awareness, see [24].

2.2 Problem Frames

The Problem Frames approach, proposed by Jackson in [13, 14], is a conceptual framework to software requirements engineering, which emphasises the need to put real software problems in their context. This contextualisation allows systematic reasoning about the required properties of the world the requirements refer to, the given properties of the world the machine must rely on, and the behavior of the machine (software) to bring about the necessary change in

the world properties. This conceptual framework has been used as a basis to (i) relate problem and solution structure [10], (ii) describe architecture-inspired problem decomposition [22], (iii) recompose software problems using a composition operator [18], and (iv) capture patterns of change in socio-technical systems [5].

Figure 2. A context diagram for a traffic lights controller.

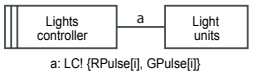
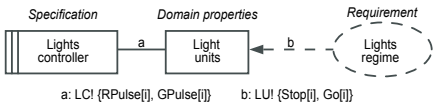


Figure 2, taken from [14], shows a context diagram for a problem in which a machine is needed to control traffic lights. The context diagram contains the Lights controller machine, marked by the rectangle with two vertical stripes, and the real world domain Light units. These domains have shared states or events (a), known as shared phenomena, which denote the actions the Lights controller can perform on the Lights units.

Figure 3. A problem diagram for a traffic lights controller.



The context diagram only describes the context in which the problem is set, and not the actual problem or the requirement, which in turn appears in the problem diagram. In figure 3, the dashed oval represents the requirement, which constrains the properties of the Light units.

Decomposition of a complex problem is done by fitting its subproblems to basic frames, for which there is a known solution. Jackson discusses five basic frames, each describing a basic class of problems (a problem pattern).

The PF approach also emphasises a clear distinction between the three different descriptions: the requirements R, the domain properties W and the specifications S. These three descriptions are linked by an entailment relationship $W, S \vdash R$. For each basic frame, the entailment relationship is expressed by a correctness argument. From the correctness argument we can derive certain frame concerns. These frame concerns help the analyst to raise questions about properties of the analysed problem, by providing a list of typical issues that apply to the problem frame at hand.

Jackson also introduces variant frames, which extend basic frames by typically adding a new domain to the problem context. A variant frame shares the central concern of the basic frame, but has additional concerns to deal with problems that do not fit basic frames. Variant frames are one technique of the PF approach considered in this paper for dealing with variability.

3 Illustrative Example

The example we use throughout this paper is a simplified sea buoy system, which was discussed by Booch in [4]. The main purpose of the sea buoy is to provide navigation and weather data to passing vessels. The buoys collect air and water temperature, wind speed, and location data through a variety of sensors.

The requirements for the basic version (p_1) of the sea buoy are as follows:

R1 Record current wind, temperature, and location information; wind speed readings are taken every 30 seconds, temperature readings every 10 seconds and location every 10 seconds.

R2 Broadcast current wind, temperature, and location information every 60 seconds.

A more sophisticated version (p_2) of the sea buoy system also provides the following additional functionality:

R3 Record and broadcast wave height measures; wave height readings are taken every 20 seconds and are broadcast together with the other gathered information.

This product line has one variation point, which is the decision as to whether the additional wave height sensor will be included or not. The following two sections discuss how the addition of the requirement R3 affects the problem structure of R1 and R2.

3.1 Feature diagram analysis

Following the guidelines proposed by Lee et al. [20], we decompose the requirements R1 and R2 as follows. Globally the buoy serves two purposes: first it gathers data (R1), then it broadcasts the gathered data (R2), making them two compulsory features. The data gathering feature can be further decomposed into three subfeatures: (i) the wind speed measurement feature, (ii) the water temperature measurement feature and (iii) the location determination feature. Features in figure 4 can be described as follows (we omit the root).

Data gathering This is a grouping feature for all data gathering activities. It has three subfeatures, corresponding to the different types of measures taken. Its cardinality

is $\{3, 3\}$, which means that all subfeatures have to be included in the product.

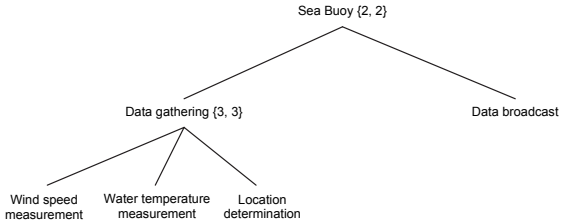
Wind speed measurement Record wind speed measurement every 30 seconds.

Water temperature measurement Record water temperature measurement every 10 seconds.

Location determination Record position information every 10 seconds.

Data broadcast Broadcast gathered weather information over the radio every 60 seconds.

Figure 4. A feature diagram for the basic sea buoy system..

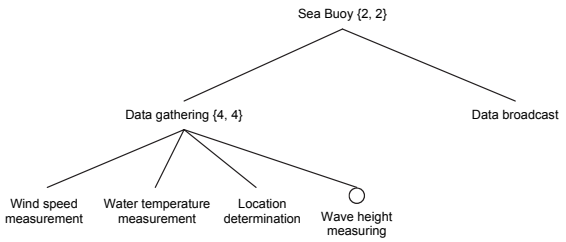


Adding the variability requirement (R3) leads to the revision of the initial diagram as shown in figure 5. In the diagram, the cardinality of the data gathering feature needs to be changed to accommodate the new branch, representing the wave height measurement feature. The new feature can be described as follows:

Wave height measurement Record wave height measurement every 20 seconds. This feature is optional, it is only included in the advanced version of the sea buoy.

It is noted that the Data broadcast feature is also affected by the addition of the new requirement.

Figure 5. A feature diagram for the basic and the advanced sea buoy system.



Given this revised feature tree, the following two products (sets of features) can be derived: $\{$*Sea Buoy, Data gathering, Data broadcast, Wind speed measurement, Water temperature measurement, location determination*$\}$ and $\{$*Sea Buoy, Data gathering, Data broadcast, Wind speed measurement, Water temperature measurement, location determination, wave height measuring*$\}$.

This is the structure of variability revealed by the FD analysis.

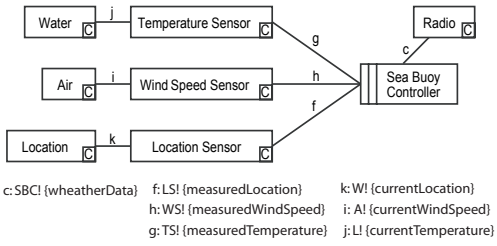
3.2 Problem analysis

We now apply a PF analysis to the same example, first using the base requirements (R1 and R2), and then adding the variability requirement (R3).

3.2.1 Analysing R1 and R2

Figure 6 shows the context diagram for R1 and R2 of the sea buoy controller. Inclusion of the three real world domains[2] (Water, Air, Location) raises concerns that otherwise would have been overlooked. These concerns are discussed towards the end of the analysis.

Figure 6. A first context diagram for the sea buoy example.

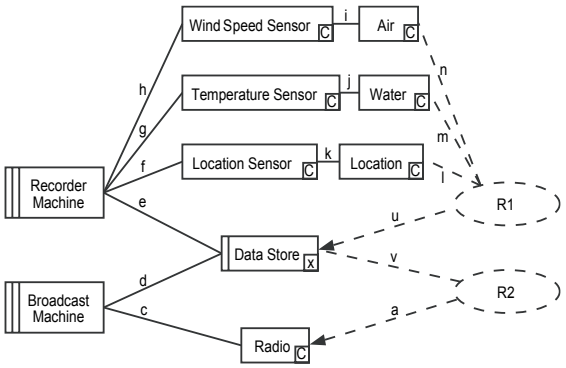


The shared phenomena between the machine and the Radio domain (c) are the emitted weather information. The Temperature Sensors domain shares the water temperature phenomenon (j) with the Water domain, and the measured water temperature (g) with the machine. The reasoning is similar for the other sensor and environmental domains. The shared phenomena descriptions will be omitted on later diagrams.

Figure 7 shows the composite problem diagram for R1 and R2. The Recorder and Broadcast Machines are a decomposition of the Sea Buoy Controller machine shown in the context diagram of figure 6. In order to link both machines, we have to introduce a new domain: the Data Store.

[2]These domains are causal domains, which is noted by the *C* in the lower right corner. Causal domains include predictable causal relationships among their causal phenomena.

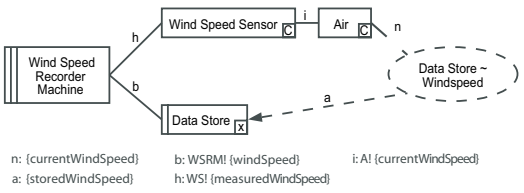
Figure 7. A composite problem diagram for the sea buoy example.



It is a designed lexical domain[3] where the measures can be stored by one machine before they are broadcast periodically by the other. The requirements are depicted as dashed ovals R1 and R2. An arrow indicates a domain that is restrained by the requirement. R1 constrains the Data Store, meaning that the data stored represents the truthful current weather information. R2 constrains the Radio, meaning that the periodic broadcast contains appropriate weather information.

The subproblem R1 is complex and requires further decomposition. We identify the following three subproblems. (R1.A) Wind speed recording subproblem: this subproblem requires that the machine reads the current wind speed and sends it to the Data Store every 30 seconds, and (R1.B) Location recording subproblem: this subproblem requires that the machine determines the current location and sends it to the Data Store every 10 seconds, and (R1.C) Temperature recording subproblem: this subproblem requires that the machine reads the current temperature. The subproblem R2 is already an instance of a basic problem frame, it requires no further decomposition.

Figure 8. The wind speed recording subproblem.



[3]Which means that it's a passive domain, where information may be stored temporarily, and that we have a certain control over its structure.

Due to space constraints, our discussion will focus on the analysis of subproblem R1.A, which is similar to the other two subproblems. Figure 8 shows the problem diagram for the subproblem R1.A, the which fits a class of problem known as Information Display[4]. The three constituent descriptions of the subproblems are given below.

Requirement. Record current wind speed information. Readings are taken every 30 seconds and written to the Data Store.

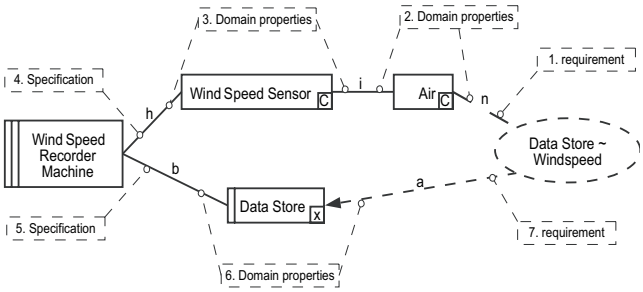
Domain properties. We assume that at any time when a given value x is visible to the sensor at interface i (see figure 8), the same value is visible to the machine at interface h at the same time, i.e. $\forall x \in \mathbb{R} \bullet x \; at \; h \Rightarrow x \; at \; i$. In other words, we assume that the water temperature sensors are working correctly. We also assume that a value sent to the Data Store at b be stored.

specification.

```
forever {
    Read x from h;
    Write x to b;
    Wait 29 sec;
}
```

Note that the specification assumes that the `read` and `write` operations are atomic, and that their execution takes exactly one second.

Figure 9. A correctness argument for subproblem R1.A.



1. Given a current wind speed, *(Requirement)*
2. because the wind speed has certain properties making it measurable,[5] *(Domain properties)*
3. the wind speed sensors are able to capture this information, *(Domain properties)*
4. the machine will then obtain the readings, *(specification)*

[4]This is one of the basic frames discussed by Jackson. The problem is to build a machine that collects information about some part of the physical world and presents it at a given place, which is the data store in our case.

[5]Common properties used by anemometers rely on the fact that there is a relation between the wind speed and the movement of flexible mechanical devices, like windmills or cups, exposed to the wind. Or the cooling-effect of the wind, which also varies with its speed.

5. and send them to the data store every 30 seconds; *(specification)*
6. the data store being capable of storing the sent information until it is needed, *(Domain properties)*
7. satisfying the requirement, that the given information should be saved every 30 seconds. *(Requirement)*

Figure 9 shows a correctness (or adequacy) argument for the subproblem R1.A explaining how the requirement, the specification and the domain properties fit together, satisfying the entailment relationship $S, W \vdash R$.

Further analysis of the problem reveals that there are important concerns to be considered.

Domain reliability concern. The assumption for the wind speed sensor domain $\forall x \in \mathbb{R} \bullet x \ at \ h \Rightarrow x \ at \ i$ is too strong, because in real life the sensor could fail or give incorrect information. This could lead to the requirement not being satisfied.

Resource shortage concern. The Data Store can only be of finite size. This means that it can indeed be full at a given moment.

Initialisation concern. The Data Store can be empty when the first broadcast is due.

These and other concerns need to be considered when implementing the specification for the subproblem.

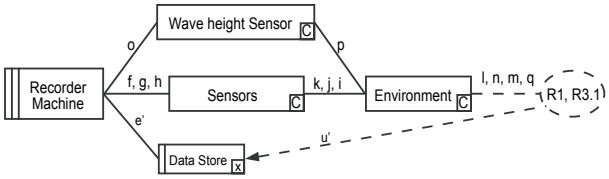
3.2.2 Adding the variability requirement.

The requirement R3, that the wave height be recorded and broadcast as well, is a complex problem that can be decomposed into two subproblems. (R3.1) Wave height recording subproblem: the machine reads the current wave height and sends it to the Data Store every 20 seconds. (R3.2) Wave height broadcast subproblem: the machine broadcasts the wave height every 60 seconds. The resulting subproblems can be considered as variants of the initial two subproblems R1 and R2, represented in figure 7.

Figure 10 shows a high level problem diagram for R1 and R3.1. The diagram has been simplified such that all sensors now form only one domain; the same holds for the environment. The Wave height Sensor domain has been added, connecting the machine to the environment, as stated by R3. The resulting problem diagram can be considered as being a variant frame of the initial diagram.

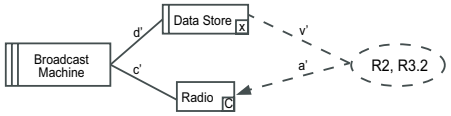
The diagram in figure 10 shows that there are no additional or changing shared phenomena between the recorder machine (respectively the environment) and the sensors. This means that a big part of the preceding problem analysis is still valid. The Sensors domain can be considered as a "black box", which is not touched by adding the Wave height Sensor domain. Hence, all concerns and all diagrams related to the sensors and the environment, are still valid.

Figure 10. A high level problem diagram for R1 and R3.1.



Other shared phenomena change. The shared phenomena e' between the Recorder Machine and the Data Store, as well as the requirement phenomena u', change as in the new problem the wave height has to be sent to the Data Store as well. A new requirement phenomenon q has been added because the requirement R1, R3.1 also refers to the wave height in the real world.

Figure 11. A high level problem diagram for R2 and R3.2.



Similar changes are observed for the broadcast subproblem of R2 and R3.2 which is represented in figure 11. In order to satisfy the new requirement, the shared phenomena between the broadcast machine and the Data Store (d') now also contain the wave height information. The same applies to the shared phenomena between the broadcast machine and the radio (c'). The requirements phenomena change as well because the requirement now also refers to the wave height communicated over the radio (a'), and the wave height in the data store (v').

These changes will raise new concerns. Due to space constraints, the correctness argument is not given here. However, it is clear that the new concerns are very similar to the ones observed in section 3.2.1. For instance, the resource shortage concern has to be reconsidered, as the Data Store has to hold more information than before. Overall, this means that the initial concerns, while still being valid, have to be revised to take into account new concerns of similar nature.

As far as the specification is concerned, it is sufficient to make sure that variability in the shared phenomena is possible. The variation barely affects the original diagram in terms of domains, and so large parts of the original analysis are still valid.

3.3 Feature diagram analysis revisited

The lessons learnt through the problem analysis are the following.

Problem structure. The optional property of the wave height feature is not a local issue, it affects other features as well. The wave height feature affects its parent, the Data Gathering feature, as well as the Data Broadcast feature. This, however, is not visible in the problem structure given by the FDs in section 3.1.

Issues and concerns. The problem analysis has revealed many issues and assumptions that could affect the specifications we write. For example, with the PF analysis it was possible to uncover domain assumptions that turned out to be too strong or unrealistic. These issues have not been discovered through the FD analysis.

Finally, the problem analysis would not complement the Feature Diagrams, if we could not refer to their features. In our case, we observe a correspondence between features and subproblems fitting basic and variant frames, as summarised in Table 1. Hence, when we are treating and discussing a feature, we can easily refer to its underlying problem structure, and to all the concerns that arise from it.

Table 1. Linking PF artefacts to features

Features	Problem frames
Data gathering	Subproblem R1, R3.1
- Wind speed measurement	- Subproblem R1.A
- Water temp. measurement	- Subproblem R1.B
- Location determination	- Subproblem R1.C
- Wave height measurement	- Subproblem R3.1
Data broadcast	Subproblem R2, R3.2
- Wind speed, water temp. and location broadcast	- Subproblem R2
- Wave height broadcast	- Subproblem R3.2

4 Related Work

Approaches to contextualising FDs already exist. Yu et al. [30] propose a procedure for translating goal models to FDs. Their early requirements analysis step uses goal models rather than problem frames. They produce preliminary FDs based on their goal models, thus making a step towards a more solution-oriented view. They also see features as being part of the solution, or at least of the system (late requirements), whereas goals represent stakeholder intentions (early requirements).

The Feature-Oriented Reuse Method (FORM) [17] is also an integrated process using FD to model variability. It extends the basic FD notation by defining a framework with guidelines and different classes of features. FORM distinguishes between capability features, operating environment features, domain technology features and implementation technique features. These four feature types are represented on a layered FD. This classification is similar to Jacksons' requirements, domain properties and specification classification. However, its aim is not to reason about the problem, it is rather aimed at structuring requirements and how they lead to solutions.

Griss et al. [9] as well as Halmans and Pohl [12] also try to put FD into context by linking them to use case diagrams. Griss et al. integrate the FODA approach [16] into the Reuse-Driven Software Engineering Business [15]. They consider use case models as being user oriented, and FD as being oriented towards the software developer. Halmans et al. use use cases to document product family variability, and to communicate it to the user.

Czarnecki and Antkiewicz [7] take a similar approach. They argue that "*features in a feature model are merely symbols*"s and that mapping them to other models "*gives them semantics*", which shows that the purpose of their work is similar to what is outlined in this paper. They describe a general approach for mapping different kinds of modelling languages to a feature model. They then illustrate it by mapping FDs to UML activity diagrams.

In a way, all these approaches aim to address the lack of context in FD, either by introducing additional goal models, activity diagrams, or additional use case diagrams. They actually observe and address the same shortcomings of FD as we do in this paper. Yet, the techniques they use to address these shortcomings do not support problem reasoning at a level of granularity that separates descriptions of requirements, domain properties, and specifications, as problem frames do. Therefore we think that our paper is a novel contribution to this discussion on problem variability.

Our approach fits also into the paradigm of *Orthogonal Variability Modelling (OVM)* introduced by Bachmann et al. [1]. This paradigm suggests the use of FDs as the central variability model, which is then linked to more detailed *base diagrams*, like class diagrams for data or state diagrams for behavior, in order to allow for more precise reasoning. Problem frames can be seen as being such a base diagram which allows detailed reasoning about the problem and its physical context. OVM suggests to extend base diagrams with explicit modelling of variability and to relate the global variability model to the variability in the base models. Although beyond the scope of this paper, the PF approach can

be extended to highlight variability in its models, and how it links to variability in the base models.

5 Conclusions & Future Work

In this paper we have examined problem-oriented approaches, and how they can be combined with well established product line development methods. We did this by analysing examples of both approaches, problem frames and FDs. Both were applied to an illustrative example, based on which we indicated different weaknesses of FDs that can be tackled by problem frames, thus showing a certain complementarity of both approaches.

In the illustrative example, we have demonstrated the benefit of an early requirements analysis using the PF approach. It allows us to understand and to reason about the problem, rather than about some set of abstract requirements. It guides us in our discovery of the problem world by offering a complete and consistent methodology. By exploring the problem in greater detail, critical issues can be discovered at a much earlier stage of the development, when there is still enough flexibility for major architectural changes.

One could argue that more elaborate dependency notations should be introduced into the FD notation, so that it can handle these problems by its own. This is done by Lee and Kang [19] as well as more recently by Zhang et al. [31]. However, since there is no notion of physical context in FDs, it is not clear how these dependencies can be detected effectively. In this sense, our approach could serve as an input for these notations.

We also managed to establish a correspondence between basic problem diagrams and features. We thus introduced a certain amount of traceability because this correspondence tells us for each feature what problem it addresses. We can associate a problem diagram with every basic feature, represent its underlying structure and analyse potential concerns. Hence, we know what the issues for a certain feature are, and what the important or critical aspects of these features will be, even before we start to implement them. However, we need to validate whether or not the link from problem frames to FDs can be made in other cases.

Finally, we have to acknowledge that FDs capture the essence, as well as the variability aspects in a very concise and intuitive way. Many different stakeholders have no difficulty talking about features, thus the same notation can be used to communicate with all of them, which simplifies things. FDs are indeed excellent at what they are intended for: represent requirements for product lines and feature-based development.

In the introduction we argued that FDs can link problem frames to the solution space. As methods like FORM [17] already link FDs to the solution space, the step left is to link problem frames to FDs. In this paper we examined a possible correspondence between artefacts of both approaches. This link has to be examined in greater detail in future work. A first step towards a formal combination of FDs and the PF approach is the definition of a formal semantics for both of them. Fortunately there has already been progress in this area: for example, Bontemps et al. [3] and Schobbens at al. [27, 25] define a semantics for FDs, and Hall et al. [11] for the PF approach. We intend to build on this work.

6 Acknowledgements

We would like to thank our colleagues at the Open University, especially Mohammed Salifu and Yijun Yu as well as Pierre-Yves Schobbens and Jean-Christophe Trigaux at the University of Namur, for their feedback and for many interesting discussions about the subject. We acknowledge the financial support of EPSRC.

References

[1] F. Bachmann, M. Goedicke, J. C. S. do Prado Leite, R. L. Nord, K. Pohl, B. Ramesh, and A. Vilbig. A meta-model for representing variability in product family development. In *PFE*, pages 66–80, 2003.

[2] D. S. Batory. Feature Models, Grammars, and Propositional Formulas. In *SPLC*, pages 7–20, 2005.

[3] Y. Bontemps, P. Heymans, P.-Y. Schobbens, and J.-C. Trigaux. Semantics of FODA Feature Diagrams. In T. Männistö and J. Bosch, editors, *Proc. of Workshop on Software Variability Management for Product Derivation Towards Tool Support held in conjunction with the 8th international Conference on Software Product Line Conference (SPLC04)*, pages 48–58, Boston, August 2004.

[4] G. Booch. Object-oriented development. *IEEE Trans. Software Eng.*, 12(2):211–221, 1986.

[5] J. Brier, L. Rapanotti, and J.Hall. Towards capturing change in socio-technical systems requirements. In *Proceedings of the 11th International Workshop on Requirements Engineering - Foundation for Software Quality*, pages 225–237, 2005.

[6] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Aug. 2001.

[7] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE*, pages 422–437, 2005.

[8] U. W. Eisenecker and K. Czarnecki. *Generative Programming: Methods, Tools, and Applications*. 2000.

[9] M. Griss, J. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB. pages 76–85, Vancouver, BC, Canada, June 1998.

[10] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 137–144. IEEE Computer Society, 2002.

[11] J. G. Hall, L. Rapanotti, and M. Jackson. Problem frame semantics for software development. *Software and System Modeling*, 4(2):189–198, 2005.

[12] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Inform., Forsch. Entwickl.*, 18(3-4):113–131, 2004.

[13] M. Jackson. *Software requirements & specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[14] M. Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[15] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse. Architecture, Process and Organization for Business Success*. 1997.

[16] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Nov. 1990.

[17] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[18] R. Laney, L. Barroca, M. Jackson, and B. Nuseibeh. Composing requirements using problem frames. In *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, pages 122–131, Washington, DC, USA, 2004.

[19] K. Lee and K. C. Kang. Feature dependency analysis for product line component design. In *ICSR*, pages 69–85, 2004.

[20] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *ICSR-7: Proceedings of the 7th International Conference on Software Reuse*, pages 62–77, London, UK, 2002. Springer-Verlag.

[21] K. Pohl, G. Bockle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, July 2005.

[22] L. Rapanotti, J. G. Hall, M. A. Jackson, and B. Nuseibeh. Architecture-driven problem decomposition. In *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, Kyoto, Japan, 2004. IEEE.

[23] M. Riebisch. Towards a More Precise Definition of Feature Models. In *In Proceedings of Modelling Variability for Object-Oriented Product Lines ECOOP Workshop, Darmstadt, Germany, July 2003*, Norderstedt, 2003. Eds. BooksOnDemand Publ. Co.

[24] M. Salifu, B. Nuseibeh, L. Rapanotti, and T. Tun. Using problem descriptions to represent context-aware variabilities. Submitted to VaMoS Workshop 2006, November 2006.

[25] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 139–148, Minneapolis, Minnesota, USA, Sept. 2006.

[26] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. Presentation for the 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis, September 2006.

[27] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks (2006), doi:10.1016/j.comnet.2006.08.008, special issue on feature interactions in emerging application domains*, page 38, 2006.

[28] A. van Deursen and P. Klint. Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[29] J. van Gurp, J. Bosch, and M. Svahnberg. On the Notion of Variability in Software Product Lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001.

[30] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. Leite. From stakeholder goals to high-variability software designs. Technical report, Computer Systems Research Institute, University of Toronto, 2005.

[31] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requir. Eng.*, 11(3):205–220, 2006.